Northwestern University
Department of Electrical Engineering and Computer Science

Programming Assignment 1

EECS 231 Advanced Programming
Fall Quarter 2006

Due : Wednesday 10/4/06 at 11:59pm

There are 5 pages to this handout

Goals of this assignment

- Introduction to C++; variables, loops, conditionals, arrays, simple I/O.
- Introduction to g++
- Introduction to makefiles

Before you begin

Before you begin, get all the files that you'll need for this assignment:

1. Log on to Wilkinson.
2. If you don't have one already, create the directory where you'll be doing your eecs231 work, and cd to it:

   ```
   mkdir 231
   cd 231
   ```

3. Now get the file from my account:

   ```
   cp ~ido715/231/pa1.tar  .
   ```

   tar files are archives, similar to Windows's zip files. Here's how to extract the contents of a tar file:

   ```
   tar xf pa1.tar
   ```

4. If you do `ls` now, you should see the directory `PA1`, which contains subdirectories `part1` and `part2`. You should also see the submission script `submit`. Do not modify that.

Part 1: Makefiles, Debugging

The directory `PA1/part1/` contains two files: `reverse.cpp` and `Makefile`.

- A `Makefile`, along with the `make` utility, provides you with a way to organize and automate compilation. It is especially useful in large projects that consist of several files. The makefile keeps track of which files have been modified since the last compilation and recompiles only those, as well as any files affected by the modifications.

  Makefiles will be discussed in more detail in class or during a lab. For now, you will only use them. To compile `reverse.cpp` using the provided makefile, type

  ```
  make
  ```

  at the command prompt. This will generate an executable called `reverse`. You can run the executable by typing

  ```
  ./reverse
  ```

- `reverse.cpp` contains a C++ program that defines an array and then is supposed to reverse its elements and print the contents of the resulting array. The original array is {1, 2, ,3 ,4, 5} and after the reversal, it is supposed to become {5, 4, 3 ,2, 1}. However, when you run the executable, you will notice that the program does not reverse the array properly. Clearly, there is a mistake ("bug") in the code.

What you should do to complete Part 1

Examine the code and try to figure out exactly what the problem is. When you do, modify the program to correct the mistake. Insert comments that explain your modification.

When you are satisfied with your answer, type

```
make clean
```

This will clear up the directory from files that you won't need any longer, namely the object and executable files that were generated.

Part 1 is now ready for submission.

Part 2: Programming, testing

Programming

Files

The directory `PA1/part2/` contains the following files:

- `reservations.cpp`: A skeleton C++ file. Write your program in it.
- `std_exe.out`: A fully working executable.
- A `Makefile` that provides you with quick ways to compile and test your code. Type `make` to compile.

Problem description

A small airline has just purchased a computer for its new automated reservations system.

You have been asked to write a program that will be used to assign seats on each flight of the airline's only plane.

You have the following seat assignment information:

- The plane has 10 seats
- Seats 1-4 are first class
- Seats 5-10 are economy class

Your program should behave as follows:

- It should display the menu:

```
Please enter 1 for "First Class".
Please enter 2 for "Economy Class".
```

- The user should be assigned a seat in the class he has requested. If that class is already full, then he should be bumped to the other class.
- When a seat has been assigned, a boarding pass should be printed:

```
**************************************************
* BOARDING PASS                                  *
* -------- ----                                  *
* SEAT NO:  1                                    *
* CLASS   : FIRST                                *
**************************************************
```

- Initially, all seats are empty.
- A seat can only be assigned exactly once (no double-booking).
- When all 10 seats have been assigned, the program should exit.

Programming issues:

- Use a one-dimensional array of size 10, to hold the seat information. Choose an appropriate type for the array elements. Write your program so that if the size of the array needs to be changed in the future (because the airline may buy a larger airplane), the code will be easy to update.
- Before you even begin to write code, design your program on paper. How will you check whether a class is fully booked? Will you count the number of seats assigned? Or will you compare the most recent assigned seat to the maximum seat number for that class? How will you keep track of the next available seat for a specific class? You may discuss the design of your program with other students, preferably via the newsgroup.
- Keep in mind that it's very possible for the program to be revised as the airline grows. For example, they may replace the current airplane with a larger one and they may add additional classes ("Business", "ScreamingChildren", etc.). Try to make design decisions that will facilitate such revisions.
- Do not try to implement all the functionality in one go. First, write a program that just prints the menu, reads and verifies the result. Then add code that assigns the next available seat, ignoring the issue of class. Make certain this works before you proceed. Finally, based on the design decisions you have made, implement the code that will assign seats based on class, as described above.

Testing

The directory `PA2/part2/` contains a sample executable, `std_exe.out` that you can try out to see how your program is expected to behave for various inputs.

The directory `PA2/part2/tests` contains a number of test cases that you can use to test your program. These are the test cases that will be used for grading. You will receive full credit if your code passes the test cases, is well-designed and sufficiently commented (see commenting guidelines below).

In order to test your program using test case X, type:

```
make testX
```

This will run the test case, save the result in file `test.X.output` and compare it to the expected output in `test.X.std`. If they match, you will see nothing. If they do not match, you will see a list of differences between the two files.

Cleaning up

As before, typing `make clean` will remove your executable (but not the official one), the object file and any coredump files that may be generated.

What you should do to complete Part 2

Write your reservations program in the provided file `reservations.cpp`. Test it and make sure it is well commented and readable (e.g. uses good variable names and is indented properly).

How to submit your assignment

Move up to directory `PA1/` and type

```
tcsh submit
```

to run the submission script. You may resubmit your assignment as many times as you need.

Getting help

Even though the reservations program is not difficult, it *is* your first C++ program. This means that you'll make many mistakes in the beginning. The program *will* take you longer to complete than you may think, so **START EARLY!**.

When you get stuck, get help. Don't spend hours trying to figure out what an error message means or why your program is giving unusual output. Post to the newsgroup or email me. The newsgroup is a better option since it will be monitored by both me and your classmates and you may get an answer faster than if you just emailed me. Just remember to never post your code (there's only one exception, described below).

If you are getting compilation errors, post the exact error message that the compiler gives you (copy and paste it). You may also post the one line of code where the error occurs.

As you know, I have no set office hours. This means that you can come by my office at any time. If you want to call in advance to find out whether I'm in, the number is 491-5708.

Commenting guidelines

These guidelines should be followed throughout the quarter. For this assignment, you only need file headers and inline comments:

- **File headers**
  Each file should have a comment at the top, containing the following:
  - A *brief* description of the file's contents
  - The author's name
  - The date the file was created
  - Revision information, if applicable.

```
// ================================================
// A driver function that tests the RaffleBox class
//
// Author: John Doe
// Date:   03/20/2004
// Revision: 01/05/2005 by Jane Smith
//            -- added checks for memory allocation
// ================================================
```

- **Class headers**
  Each class should be preceded by a comment containing the following:
  - The class name
  - The classes it is derived from (if applicable)
  - A *brief* description of the class
  - Revision information (if applicable)

- **Function headers**
  Each function should be preceded by a comment containing the following:
  - The function name
  - Its inputs and the role of each input value
  - What the function accomplishes and returns
  - Any preconditions or assumptions about the input values
  - Any postconditions

- **Inline comments**
  Add inline comments as necessary. Note that inline comments should never restate the code (e.g. increment i) but instead explain what it does (e.g. add a ticket to the raffle box).